Dynamic Shortest Path Algorithms for Hypergraphs

Jianhang Gao, Qing Zhao, *Fellow, IEEE*, Wei Ren, Ananthram Swami, *Fellow, IEEE*, Ram Ramanathan, *Fellow, IEEE*, and Amotz Bar-Noy

***Abstract*—A hypergraph is a set V of vertices and a set of nonempty subsets of V, called hyperedges. Unlike graphs, hypergraphs can capture higher-order interactions in social and communication networks that go beyond a simple union of pairwise relationships. In this paper, we consider the shortest path problem in hypergraphs. We develop two algorithms for finding and maintaining the shortest hyperpaths in a dynamic network with both weight and topological changes. These two algorithms are the first to address the fully dynamic shortest path problem in a general hypergraph. They complement each other by partitioning the application space based on the nature of the change dynamics and the type of the hypergraph. We analyze the time complexity of the proposed algorithms and perform simulation experiments for random geometric hypergraphs, energy efficient routing in multichannel multiradio networks, and the Enron email data set. The experiment with the Enron email data set illustrates the application of the proposed algorithms in social networks for identifying the most important actor and the latent social relationship based on the closeness centrality metric.**

***Index Terms*—Dynamic, hypergraph, hyperpath, shortest path.**

I. Introduction

A GRAPH is a mathematical abstraction for modeling networks, in which nodes are represented by vertices and pairwise relationships by edges between vertices. A graph is thus given by a vertex set V and an edge set E consisting of cardinality-2 subsets of V. A hypergraph is a natural extension of a graph obtained by removing the constraint on the cardinality of an edge: Any nonempty subset of V can be an element (a hyperedge) of the edge set E (see Fig. 1). It thus captures group behaviors and higher-dimensional relationships in complex networks that are more than a simple union of pairwise relationships. Examples include communities and collaboration teams in social networks, document clusters in information networks, and cliques, neighborhoods, and multicast groups in communication networks.

Manuscript received September 16, 2013; revised June 04, 2014; accepted July 14, 2014; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Ramasubramanian. This work was supported by the Army Research Laboratory Network Science CTA under Cooperative Agreement W911NF-09-2-0053.

J. Gao and Q. Zhao are with the Department of Electrical and Computer Engineering, University of California, Davis, Davis, CA 95618 USA (e-mail: jhgao@ucdavis.edu).

W. Ren is with the Microsoft Corperation, Redmond, WA 98052 USA.

A. Swami is with the Army Research Laboratory, Adelphi, MD 20783 USA.

R. Ramanathan is with Raytheon BBN Technologies, Cambridge, MA 02138 USA.

A. Bar-Noy is with the Department of Computer Science, City University of New York, Brooklyn, NY 11210 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2014.2343914

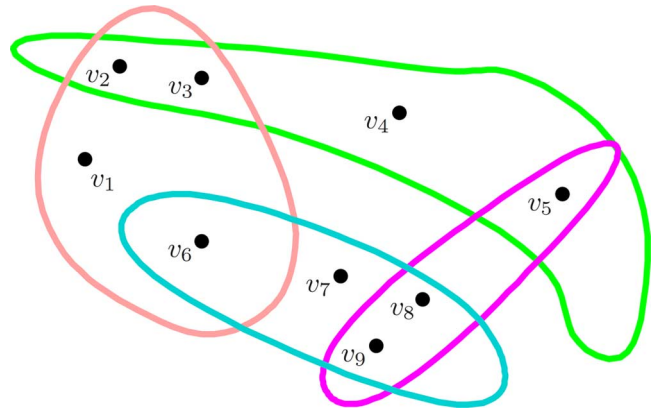

Fig. 1. Example hypergraph with four hyperedges: (v_1, v_2, v_3, v_6), (v_2, v_3, v_4, v_5), (v_6, v_7, v_8, v_9), and (v_5, v_8, v_9).

While the concept of hypergraph has been around since the 1920s (see, for example, [1]), many well-solved algorithmic problems in graph remain largely open under this more general model. Here, we address the shortest path problem.

A. Shortest Path Problem in Graphs

The shortest path problem is perhaps one of the most basic problems in graph theory. It asks for the shortest path between two vertices or from a source vertex to all the other vertices (i.e., the single-source version or the shortest path tree). Depending on whether the edge weights can be negative, the problem can be solved via Dijkstra's algorithm or the Bellman–Ford algorithm.

The dynamic version of the shortest path problem is to maintain the shortest path tree without recomputing from scratch during a sequence of changes to the graph. A typical change to a graph includes weight increase, weight decrease, edge insertion, and edge deletion. The last two types of changes model network topological changes, but they can be conceptually considered as special cases of weight changes by allowing infinite edge weights. Thus, if the sequence of changes contains only edge deletion and weight increase, we call it a decremental problem; if it contains only edge insertion and weight decrease, we call it an incremental problem. Otherwise, we have a fully dynamic problem. If multiple edges can change simultaneously, then it is a batch problem. Example dynamic shortest path algorithms for graphs can be found in [2]–[5].

B. Shortest Path Problem in Hypergraphs

Both the static and dynamic shortest path problems have a corresponding version in hypergraphs. A hyperpath in a hypergraph is a sequence of hyperedges with two adjacent hyperedges sharing at least one vertices. The weight of a hyperpath is the sum of the weights of each hyperedge on this hyperpath. The

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2014		**00-00-2014 to 00-00-2014**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Dynamic Shortest Path Algorithms for Hypergraphs	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Davis,Department of Electrical and Computer Engineering,Davis,CA,95616**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

A hypergraph is a set of vertices and a set of nonempty subsets of , called hyperedges. Unlike graphs, hypergraphs can capture higher-order interactions in social and communication networks that go beyond a simple union of pairwise relationships. In this paper, we consider the shortest path problem in hypergraphs. We develop two algorithms for finding and maintaining the shortest hyperpaths in a dynamic network with both weight and topological changes. These two algorithms are the first to address the fully dynamic shortest path problem in a general hypergraph. They complement each other by partitioning the application space based on the nature of the change dynamics and the type of the hypergraph. We analyze the time complexity of the proposed algorithms and perform simulation experiments for random geometric hypergraphs, energy efficient routing inmultichannel multiradio networks, and the Enron email data set. The experiment with the Enron email data set illustrates the application of the proposed algorithms in social networks for identifying the most important actor and the latent social relationship based on the closeness centrality metric.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**13**	

shortest hyperpath problem in weighted hypergraphs where a nonnegative weight is attached to each hyperedge is thus clearly defined. The static shortest hyperpath problem was considered by Gallo *et al.* [6] in which Dijkstra's algorithm was extended to obtain the shortest hyperpaths. Ausiello *et al.* proposed a semi-dynamic minimum length hyperpath algorithm with time complexity $O(K|H|\log h+Q)$ for directed hypergraphs. While they considered a more general definition of hyperpath length (which can be an arbitrary function of the hyperedge weight on the hyperpath), their dynamic algorithm was developed under a very restrictive assumption. Specifically, it was assumed that the length of a path can only come from a finite discrete set with size K. Since the time complexity of their algorithm is linear in K, their algorithm is not applicable to hypergraphs where edge weights can be any real number as considered in this paper. Furthermore, they considered only the incremental part of the dynamic maintenance, while in this paper we consider both incremental and decremental problems. A dynamic algorithm for the batch problem for a special class of hypergraphs was developed in [3].

With the exception of the above few studies, the shortest hyperpath problem remains largely unexplored. To the best of our knowledge, no algorithms exist for the fully dynamic problem in a general hypergraph.

In this paper, we develop two fully dynamic shortest path algorithms for general hypergraphs. These two algorithms complement each other, with each preferred in different types of hypergraphs and change dynamics.

Referred to as the HyperEdge-based Dynamic Shortest Path algorithm (HE-DSP), the first algorithm is an extension of the dynamic Dijkstra's algorithm for graphs to hypergraphs (parallel to Gallo's extension of the static Dijkstra's algorithm to hypergraphs in [6]). The extension of the dynamic Dijkstra's algorithm to hypergraphs is more involved than that of the static Dijkstra's algorithm. This is due to the loss of the tree structure (in the original graph sense) in the collection of the shortest hyperpaths from a source to all other vertices. Since the dynamic Dijkstra's algorithm relies on the tree structure to update the shortest paths after an incremental change, special care must be taken when extending it to hypergraphs.

The second algorithm is rooted in the idea of Dimension Reduction and is referred to as DR-DSP. The basic idea is to reduce the problem to that in an induced graph derived from the original hypergraph. The induced graph of a hypergraph has the same vertex set and has an edge between two vertices if and only if there is at least one hyperedge containing these two vertices in the original hypergraph. The weight of an edge in the induced graph is defined as the minimum weight among all hyperedges containing the two vertices of this edge. The shortest hyperpath in the hypergraph can thus be obtained from the shortest path in the induced graph by substituting each edge along the shortest path with the hyperedge that lent its weight to this edge. The correctness and advantage of this algorithm are readily seen: The definition of weight in the induced graph captures the minimum cost offered by all hyperedges in choosing a path between two vertices, thus ensuring the correctness of the algorithm; the reduction of a hypergraph to its induced graph removes many hyperedges from consideration when finding the shortest path, leading to efficiency and agility to dynamic changes.

As will be shown in the time complexity analysis given in Section V, HE-DSP is more efficient in hypergraphs that are densely connected through high-dimensional hyperedges and for network dynamics where changes often occur to hyperedges that are not on the current shortest hyperpaths. DR-DSP has lower complexity when hyperedge changes often lead to changes in the shortest hyperpaths. This is usually the case in networks where hyperedges in the shortest hyperpaths are more prone to changes due to attacks, frequent use, or higher priority in maintenance and upgrade. While we focus on undirected hypergraphs in the paper, the two algorithms apply to directed hypergraphs as discussed in Section VI. They also apply to batch hyperedge change problems with minor modifications.

C. Applications

Shortest path computations on hypergraphs can be applied to communication as well as social networks. An example application is routing in multichannel multiradio ad hoc networks [8]. As detailed in Section VII, we consider energy-efficient routing where the link metric is energy consumption including both transmitting and receiving energy. To capture the presence of different neighbor sets associated with different channels available to a particular node, a directed hypergraph model is needed in which a hyperedge is formed from a vertex to its neighbors that shares a particular channel and different channels available to this vertex lead to different hyperedges. The weight of a hyperedge is given by the total energy cost of a transmission from the sending vertex using the channel corresponding to this hyperedge. A shortest hyperpath from the source to the destination is the least-energy-cost route, and as network topology changes, a dynamic algorithm is required to maintain the shortest hyperpath.

In social networks, information (results, event reports, opinions, rumors, etc.) propagates through diverse communication means including direct links (e.g., gestures, optical, satcom, regular phone call, e-mail), social media (e.g., Facebook, Twitter, blogs), mailing lists, and newsgroups. Such a network may be modeled as a hypergraph with the weight of a hyperedge reflecting the cost, credibility, and/or delay in disseminating information among all vertices of this hyperedge. In particular, the weight of a hyperedge can capture the unique effect on the information after it passes through a group of people. For instance, a result can be discussed by overlapping blog collaboration networks as it spreads, and often the discussion yields a better result than if it only spreads through individuals. The minimum-cost information-passing in social networks can thus be modeled as a shortest hyperpath problem with suitably defined weights.

Another potential application is that of finding the most important actor and the latent relationships in a social network. Under a graph model of social networks, the relative importance of a vertex can be measured by its betweenness and closeness centrality indices. The former is defined based on the number of shortest paths that pass through this vertex, and the latter, the total weight of the shortest paths from this vertex to all the other vertices [13]. In a social network exhibiting hyper-relationships, betweenness and closeness centrality, based on the shortest hyperpaths, would be better indicators of the relative importance

of each actor. In Section VII, we apply the proposed shortest hyperpath algorithms to the Enron e-mail data set. We propose a weight function that leads to the successful identification of the CEO of Enron as the most important actor under the closeness centrality metric. The distance of each person in the data set to the CEO along the resulting shortest hyperpaths closely reflects the hierarchical structure of the company.

II. Dynamic Shortest Hyperpath Problem

In this seciton, we introduce some basic concepts of hypergraph and define the static and the dynamic shortest hyperpath problems. Some basic properties of the shortest hyperpaths are established and will be used in developing the dynamic algorithms in subsequent sections.

A. Hypergraph and Hyperpath

Let V be a finite set and E a family of subsets of V. If for all elements $e_i \in E$, the following conditions are satisfied:

$$e_i \neq \emptyset, \qquad \cup_{e_i \in E}\, e_i = V$$

then the couple $H = (V, E)$ is called a *(undirected) hypergraph*. Each element $v \in V$ is called a *vertex*, and each element $e \in E$ a *hyperedge*.

A *weighted undirected hypergraph* is a triple $H = (V, E, w)$ with $w : E \rightarrow \{R^+ \cup \{0\}\}$ being a nonnegative weight function defined for each hyperedge in E.

In a hypergraph, a hyperpath is defined as follows.

Definition 1: A *hyperpath* between two vertices u and v is a sequence of hyperedges $\{e_0, e_1, \ldots, e_m\}$ such that $u \in e_0$, $v \in e_m$, and $e_i \cap e_{i+1} \neq \emptyset$ for $i = 0, \ldots, m-1$. A hyperpath is *simple* if nonadjacent hyperedges in the path are nonoverlapping, *i.e.,* $e_i \cap e_j = \emptyset, \forall j \notin \{i, i \pm 1\}$.

Let $L_e = \{e_0, \ldots, e_m\}$ be a hyperpath in a weighted hypergraph H. We define the weight of L_e as

$$w(L_e) = \sum_{i=0}^{m} w(e_i).$$

B. Shortest Hyperpath and Relationship Tree

Given two vertices u and v, a natural question is to find the shortest hyperpath (in terms of the path weight) from u to v. Since the weight function is nonnegative, it suffices to consider only simple hyperpaths. If the shortest hyperpath is not simple, we can always generate a simple hyperpath without increasing the weight by deleting all the hyperedges between two overlapping nonadjacent hyperedges.

The dynamic shortest hyperpath problem can be similarly defined for a sequence $C = \{\delta_1, \delta_2, \ldots, \delta_l\}$ of hyperedge changes. Hyperedge changes are of the same four types as edge changes in a graph: weight increase, weight decrease, hyperedge insertion, and hyperedge deletion. Similarly, weight increase and hyperedge deletion will be treated together, so will weight decrease and hyperedge insertion.

In this paper, we consider the single-source shortest hyperpath problem: Find the shortest hyperpaths from a given source s to all other vertices. We focus on undirected hypergraphs first and defer the treatment of directed hypergraphs to Section VI.

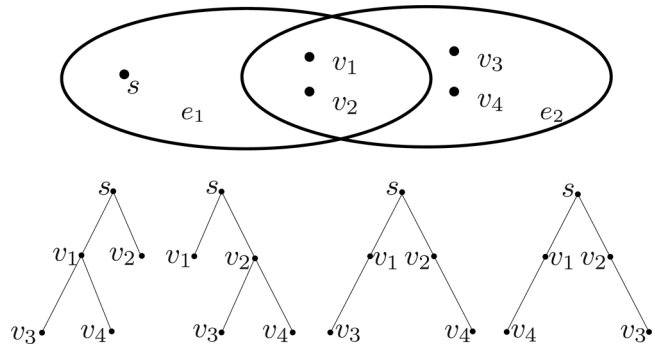

Fig. 2. Hyperpaths and the associated relationship trees.

We establish a basic property of shortest hyperpaths. It states that any segment of a shortest hyperpath is also a shortest hyperpath between any two vertices with each in one of the two end hyperedges of this segment.

Lemma 1: Let $L = \{e_1, e_2, \ldots, e_l\}$ be a shortest hyperpath from $s \in e_1$ to $z \in e_l$. Then, for any vertex $v \in e_i \cap e_{i+1}$, the hyperpath $L_v = \{e_1, e_2, \ldots, e_i\}$ is a shortest hyperpath from s to v. Furthermore, for any two vertices $u, v \in e_i \cap e_{i+1}$ (if there exist at least two vertices in $e_i \cap e_{i+1}$), the shortest distances from s to both vertices are equal.

Proof: We will prove by contradiction. Assume that $L_v = \{e_1, e_2, \ldots, e_i\}$ is not a shortest hyperpath for v. Then, there exists a different hyperpath $L'_v = \{e'_1, e'_2, \ldots, e'_k\}$ with $w(L'_v) < w(L_v)$. Then, consider the hyperpath $L' = \{e'_1, e'_2, \ldots, e'_k, e_{i+1}, e_{i+2}, \ldots, e_l\}$, we have $w(L') < w(L)$, which contradicts the fact that L is a shortest hyperpath to z. This completes the proof for the first part of the lemma. Furthermore, for any two nodes $u, v \in e_i \cap e_{i+1}$, since L_v is the shortest hyperpath for both vertices, the shortest distances from s to both vertices equal to $w(L_v)$. ■

Next, we introduce the concept of relationship tree that is needed in the proposed dynamic shortest hyperpath algorithm HE-DSP. Since two adjacent hyperedges in a hyperpath may overlap at more than one vertex, the shortest hyperpaths from s to all other vertices do not generally form a tree in the original graph sense. For the development of the dynamic shortest hyperpath algorithms, we introduce the concept of *relationship tree* to indicate the parent–child relationship along shortest hyperpaths. The concept can be easily explained in the example given in Fig. 2. Let $\{e_1, e_2\}$ be a shortest hyperpath from s to v_4. By Lemma 1, $\{e_1\}$ is a shortest hyperpath for both v_1 and v_2. As illustrated in Fig. 2, there are four possible relationship trees to indicate the parent–child relationship in these shortest hyperpaths. We will show in Section III that the choice of the relationship tree does not affect the correctness or performance of the proposed algorithm HE-DSP.

C. Notations

In the following, $D[v]$ denotes the distance of a vertex v to the source s on the shortest hyperpath, $P[v]$ the parent of v in the chosen relationship tree associated with the shortest hyperpaths,

and $E[v]$ the hyperedge containing v and $P[v]$ on the shortest hyperpath (i.e., the hyperedge that leads to v from $P[v]$ on the shortest hyperpath).

A vertex v is called an affected vertex if $D[v]$, $P[v]$, and/or $E[v]$ change in the new set of shortest hyperpaths. A hyperedge is called an affected edge if it contains an affected vertex. When it is necessary to distinguish the shortest distance before and after a weight change, $d[v]$ will denote the shortest distance before the change, $d'[v]$ the shortest distance after the change, and $D[v]$ the actual value stored in the data structure during the execution of the algorithm.

III. Hyperedge-Based Dynamic Shortest Path (HE-DSP) Algorithm

In this section, we propose HE-DSP. It is an extension of the dynamic Dijkstra's algorithm to hypergraphs. The extension is more complex than Gallo's extension of the static Dijkstra's algorithm since the dynamic Dijkstra's algorithm for graphs relies on the tree structure of the shortest paths, a structure no longer there for the shortest hyperpaths.

A. Hyperedge Weight Decrease

For weight decrease, consider that the weight of a hyperedge e decreases to w_{new}. It can be shown that the vertex $u \in e$ with $D[u] = \min_{v \in e}\{D[v]\}$ will not be affected. We then check whether the other vertices in e are affected by checking whether the following inequality holds:

$$D[u] + w_{\text{new}} < D[v]. \tag{1}$$

We then put all the affected vertices into a priority queue[1] Q. The rest of the procedure is similar to the dynamic Dijkstra's algorithm, except that when we update the distance of a vertex, we check all the hyperedges that contain this vertex. A detailed implementation of the algorithm is given as follows.

HE-DSP: Weight Decrease $(\check{e}, w_{\text{new}})$.
Step0 (Update the hypergraph)
$w(\check{e}) \leftarrow w_{\text{new}}$
Step1 (Determine the affected vertices in e**)**
$x \leftarrow \arg\min_{v \in \check{e}}\{D[v]\}$
for each $v \in \check{e}$ such that $D[x] + w_{\text{new}} < D[v]$ **do**
  $D[v] \leftarrow D[x] + w_{\text{new}}; P[v] \leftarrow x; E[v] \leftarrow \check{e}$
  $\text{Enqueue}(Q, \langle v, D[v]\rangle)$
end
Step2 (Iteratively enqueue and update affected vertices)
while NonEmpty (Q) **do**
  $\langle z, D[z]\rangle \leftarrow \text{Dequeue}(Q)$
  for each $e \in E$ s.t. $z \in e$
    for each $v \in e$
      if $D[v] > D[z] + w(e)$ **then**
        $D[v] \leftarrow D(z) + w(e); P[v] \leftarrow z; E[v] \leftarrow e$
        Enqueue or $\text{Update}(Q, \langle v, D[v]\rangle)$
end; end; end; end

[1]A priority queue is an abstract data type with the following access protocol: Only the highest-priority element can be accessed. Basic operations of a priority queue include Enqueue (add a new item to the queue), Dequeue (remove the item with the highest priority and return this item), Update (change the priority of one item in the queue), and Peek (obtain the value of the item with the highest priority). Standard implementations of a priority queue with different time complexities include array, link list, Binary heap, and Fibonacci heap [14].

The following theorem states the correctness of the algorithm.

Theorem 1: If before the weight decrease, $D[v] = d[v]$, $E[v]$, and $P[v]$ are correct for all $v \in V$, then after the weight decrease, $D[v] = d'[v]$, and $E[v]$ and $P[v]$ are correctly updated by HE-DSP.

Proof: See Appendix A. ∎

B. Hyperedge Weight Increase

To handle hyperedge weight increase, special care must be taken to find all the affected vertices. This process in the graph case relies on the tree structure of the shortest paths that no longer exists in shortest hyperpaths. Our solution is to use a *relationship tree* as introduced in Section II-B.

Consider that the weight of a hyperedge e increases to w_{new}. If e is not an edge in any of the current shortest hyperpaths, then none of the vertices will be affected; all shortest hyperpaths remain unchanged. Otherwise, the descendants, and only the descendants of this edge in the current relationship tree, may be affected. For these vertices, some of them will have increased distances, some of them will go through an alternative path with the same distance (but changed parent or hyperedge), while the rest will not be affected. In order to classify the vertices into these three categories, we propose the following coloring process, which is a modification of the coloring idea in Frigioni's algorithm for graphs [4] to take care of the nonunique choice of parent in hypergraphs.

1) v is colored *white* if $d'[v] = d[v]$ while keeping the current $P[v]$ and $E[v]$.
2) v is colored *pink* if $d'[v] = d[v]$, but only possible through a new $P[v]$ or $E[v]$ or both.
3) v is colored *red* if $d'[v] < d[v]$.

It can be shown that if a vertex x is white or pink, all its descendants in the relationship tree are white; if x is red, all its descendants are either red or pink. Therefore, we have the following coloring procedure. We first determine the color of each vertex in e. Specifically, the vertex $u \in e$ with $D[u] = \min_{v \in e}\{D[v]\}$ will not be affected and will be colored white. Any other vertex (say v) in e will be either pink or red, which we can determine by checking whether there is an alternative shortest hyperpath with the same distance for v (note that v cannot be white due to the weight change of hyperedge e that is on its current shortest hyperpath); if such a path exists, then we color v pink, otherwise we color it red and put all its children in a priority queue M. The procedure then iterates over each vertex in M according to an increasing order of the vertex distances.

After the coloring process, we only need to deal with the red vertices. For each red vertex z, we initialize its distance with the distance of the shortest path through one of its nonred neighbors and put z in another priority queue Q (if no nonred neighbor exists, we initialize it with ∞). We then iterate by extracting the vertex at the top of Q and updating its neighbors and Q until Q is empty.

A detailed implementation of the algorithm is given as follows.

HE-DSP: Weight Increase $(\check{e}, w_{\text{new}})$.
Step0 (Update the hypergraph)
$w(\check{e}) \leftarrow w_{\text{new}}$
Step1 (Determine the affected vertices in e**)**
for each $v \in \check{e}$ s.t. $E[v] = \check{e}$ **do**
Enqueue$(M, \langle v, D[v]\rangle)$
Step2 (Coloring process)
while NonEmpty(M)
$\langle z, D[z]\rangle \leftarrow$ Dequeue(M)
if $\exists$ *nonred* $q \in V$ s.t. $\exists e \in E$ with $q, z \in e$ and $D[q] + w(e) = D[z]$
then z is pink; $P[z] = q$; $E[z] = e$;
else z is red; Enqueue(M, all z's children)
end; end
Step3.a (Initialize the distance vector for red vertices)
for each *red* vertex z **do**
if z has no *nonred* neighbor
then $D[z] \leftarrow +\infty$; $P[z] \leftarrow$ Null
else
let u be the *best nonred neighbor* of z
$E[z] \leftarrow \arg\min_{e \in E, e \ni u,z}\{w(e)\}$;
$D[z] \leftarrow D[u] + w(E[z])$; $P[z] \leftarrow u$;
Enqueue$(Q, \langle z, D[z]\rangle)$
end; end; end
Step3.b: Step2 of HE-DSP: Weight Decrease

The theorem below states the correctness of the algorithm.

Theorem 2: If before the weight increase, $D[v] = d[v]$, $E[v]$ and $P[v]$ are correct for all $v \in V$, then after the weight increase, $D[v] = d'[v]$, and also $E[v]$ and $P[v]$ are correctly updated by HE-DSP regardless of the choice of the relationship tree.

Proof: See Appendix B. ∎

IV. Dimension-Reduction-Based Dynamic Shortest Path (DR-DSP) Algorithm

In this section, we propose DR-DSP. DR-DSP reduces the single hyperedge change problem in a hypergraph to a batch problem in an underlying graph. When the dynamic problem degenerates to the static problem, DR-DSP leads to an alternative algorithm for solving the static shortest hyperpath problem.

A. Static Case: DR-SP

We first consider the static version of the algorithm (referred to as DR-SP), which captures the basic idea of dimension reduction. We introduce an underlying graph that captures the essential (rather than complete) information for shortest path calculation with only n vertices (see Section VIII for a comparison of our approach and other graph transformation schemes).

The proposed DR-SP algorithm is based on the following theorem in which we show that for a general hypergraph H, the weight $\omega(L^*)$ of the shortest path L^* of H is equal to the weight of the shortest path L_G^* of an induced graph G derived from H. Specifically, corresponding to every hyperedge e in H, G contains a clique defined on the vertices of e.

Theorem 3: Let $H = (V, E, w)$ be a hypergraph, and $G = (V, \tilde{E})$ the induced graph of H where an edge $\tilde{e} \in \tilde{E}$ if and only if $\exists e \in E$ such that $\tilde{e} \subset e$. For each edge $\tilde{e}$ in G, its weight $w_G(\tilde{e})$ is defined as

$$w_G(\tilde{e}) = \min_{\{e \in E : e \supseteq \tilde{e}\}} w(e). \tag{2}$$

Let L^* and L_G^* be the shortest paths from $u \in V$ to $v \in V$ in H and G, respectively. Then, we have that

$$w(L^*) = w_G(L_G^*).$$

Proof: First, for each shortest path L_G^* in G, we can obtain a corresponding hyperpath L in H with the same weight based on (2), therefore we have that

$$w_G(L_G^*) = w(L) \geq w(L^*).$$

Then, it suffices to show that there exists a path L_G in G such that $w_G(L_G) \leq w(L^*)$, which implies that $w_G(L_G^*) \leq w_G(L_G) \leq w(L^*)$.

Assume that $L^* = \{e_0, e_1, \ldots, e_{k-1}\}$ is a shortest hyperedge path from v_0 to v_k in H where $v_0 \in e_0$ and $v_k \in e_{k-1}$. Let $v_i \in e_{i-1} \cap e_i$ $(i = 1, 2, \ldots, k-1)$ be one of the vertices in the intersection of hyperedges e_{i-1} and e_i. Construct a path $L_G = \{v_0, v_1, \ldots, v_k\}$ in the graph G. For each edge $\tilde{e}_i = \{v_i, v_{i+1}\}$ $(i = 0, 1, \ldots, k-1)$, since $\tilde{e}_i \subseteq e_i$, it follows from (2) that

$$w_G(\tilde{e}_i) \leq w(e_i).$$

Thus

$$w_G(L_G) = \sum_{i=0}^{k-1} w_G(\tilde{e}_i) \leq \sum_{i=0}^{k-1} w(e_i) = w(L^*)$$

i.e., $w_G(L_G) \leq w(L^*)$. ∎

It follows from Theorem 3 that the shortest path in a general hypergraph can be obtained by applying Dijkstra's algorithm to the induced graph G as defined in the theorem.

B. Dynamic Case: DR-DSP

We now consider the dynamic case of the problem. Since the underlying graph introduced in Section IV-A does not capture the complete information of the original hypergraph, the key challenge here is how to update the underlying graph with polynomial time complexity when changes occur in the original hypergraph. Our approach is to introduce a special data structure—a list of priority queues—to maintain the complete information of the original hypergraph.

In the dynamic case, a sequence $C = \{\delta_1, \delta_2, \ldots, \delta_l\}$ of hyperedge changes in the hypergraph H results in a sequence of edge changes in the induced graph G. For each hyperedge change δ_i, DR-DSP first updates the induced graph G to locate all the changed edges caused by δ_i. In the next step, DR-DSP updates the shortest path tree in the induced graph G.

Consider first the graph update. A change to a hyperedge e only affects those edges in G that are subsets of e, i.e., a hyperedge change is localized in the induced graph G. Furthermore, since the weight of an edge in G is the minimum weight

of all hyperedges containing it, not all edges in G that are subsets of e will change weight. Based on these observations, we propose a special data structure and procedure for updating the induced graph G without regenerating the graph from scratch. Specifically, at the initialization stage of the algorithm, a priority queue M_{uv} for each pair of vertices (u, v) in the hypergraph is established to store the weights of all hyperedges that contain both u and v. When a change occurs to hyperedge e, all the priority queues M_{uv} associated with the pair of vertices (u, v) that are contained in e are updated with the new weight of e. Thus, the top of these priority queues always maintains the weight for edge (u, v) in the induced graph G for each (u, v). A detailed implementation of the proposed procedure is given as follows.

Graph Update $\check{e}w_{\text{new}}$
for each $u, v \in \check{e}$
  $\text{Update}(M_{uv}, < \check{e}, w_{\text{new}} >)$;
  $w_{uv} \leftarrow \text{Peek}(M_{uv})$;
end;

After the induced graph G has been updated, we now face a dynamic shortest path problem in a graph. However, since a single hyperedge change can result in multiple edge changes in G, we need to handle a batch problem. While existing batch algorithms and iterative single-change algorithms for graphs can be directly applied here, we show that the batch problem we have at hand has two unique properties that can be exploited to improve the efficiency of the algorithm.

Property 1: The edge changes in G caused by a hyperedge change are either all weight decreases or all weight increases.

Property 2: All changed edges in G caused by a hyperedge change belong to a clique in G.

In the case of weight decrease, if the weight of hyperedge e decreases to w_{new} , by Theorem 3 and Property 1, there are (possibly) several edge-weight decreases in the induced graph G. We know that there is at least one unaffected node $x = \arg\min_{v \in e}\{D[v]\}$. By Property 2, these affected edges are contained in a clique derived from the changed hyperedge; therefore, it is sufficient to determine the distance of every node v (other than x) in the original changed hyperedge e by checking whether $D[x] + w_{\text{new}} < D[v]$. We can initialize the priority queue with those nodes whose weights decrease. After that, the procedure is similar to dynamic Dijkstra's algorithm in the graph case. Detailed implementations of DR-DSP weight decrease are given as follows.

DR-DSP: Weight Decrease $(\check{e}, w_{\text{new}})$.
Step0 (Update the hypergraph and *G*)
  $w(\check{e}) \leftarrow w_{\text{new}}$
  Graph Update $(\check{e}, w_{\text{new}})$
Step1 of HE-DSP: Weight Decrease
Step2 (Iteratively update all affected vertices)
  while NonEmpty (Q) **do**
    $\langle z, D[z]\rangle \leftarrow \text{Dequeue}(Q)$
    for each $v \in V$ s.t. $(z, v) \in E$
      if $D[v] > D[z] + w(z, v)$ **then**
        $D[v] \leftarrow D(z) + w(z, v); P[v] \leftarrow z$
        Enqueue or $\text{Update}(Q, \langle v, D[v]\rangle)$
end; end; end

In the case of weight incresae, if the weight of hyperedge e increases to w_{new}, by Theorem 3 and Property 1, there are (possibly) several edge-weight increases in the induced graph G. Similar to the single-change case in graph, there is at least one unaffected node $x = \arg\min_{v \in e}\{D[v]\}$. Then, another node $v \in e$ is affected only if $E[v] = e$, i.e., e is on its shortest hyperpath. We use all such nodes to initialize the priority queue M. The rest is similar to that in the dynamic Dijkstra's algorithm case.

Detailed implementations of DR-DSP weight increase are given as follows.

DR-DSP: Weight $(\check{e}, w_{\text{new}})$.
Step0 (Update the hypergraph and *G*)
  $w(\check{e}) \leftarrow w_{\text{new}}$
  Graph Update $(\check{e}, w_{\text{new}})$
Step1 of HE-DSP: Weight Increase
Step 2 (Coloring Process)
  while $\text{NonEmpty}(M)$
    $\langle z, D[z]\rangle \leftarrow \text{Dequeue}(M)$
    if $\exists$ *nonred* $q \in V$ s.t. $D[q] + w(q, z) = D[z]$
      then z is pink
      else z is red; Enqueue(M, all z's children)
    end; end
Step3.a (Initialize the distance vector for red vertices)
  for each *red* vertex z **do**
    if z has no *nonred* neighbor
      then $D[z] \leftarrow +\infty$; $P[z] \leftarrow$ Null
      else
        let u be the *best nonred neighbor* of z
        $D[z] \leftarrow D[u] + w(u, z)$; $P[z] \leftarrow u$
        $\text{Enqueue}(Q, \langle z, D[z]\rangle)$
      end; end; end
Step3.b: Step2 of DR-DSP: Weight Decrease

V. Time Complexity Analysis

Given a hypergraph $H = (V, E, w)$ and a change to hyperedge $\check{e}$, let $|\delta|$ denote the number of affected vertices, $|\delta_\Phi| = \sum_{e \in E, e \text{ is affected}} |e|^2$, m the number of edges in the induced graph, and $\|\tilde{\delta}\|$ the number of affected edges in the induced graph plus $|\delta|$.

Theorem 4: The time complexities of HE-DSP and DR-DSP are $O(|\delta| \log |\delta| + |\delta_\Phi|)$ and $O(|\delta| \log |\delta| + \|\tilde{\delta}\| + |\check{e}|^2 \log m)$, respectively.

Proof: The time complexity of DR-SP mainly comes from Steps 1 and 2. Step 2 is essentially applying Dijkstra's algorithm to a graph with n vertices and $\tilde{m}$ edges where $\tilde{m}$ is the number of edges in the induced graph G. The running time is thus $O(n \log n + \tilde{m})$. An implementation of Step 1 is to obtain the edge weight $w_G(\tilde{e})$ based on (2). Therefore, the time complexity for Step 1 is $O(\sum_{e \in E} |e|^2)$, *i.e.,* $O(\Phi)$. With $\tilde{m}$ upper-bounded by Φ (since for each $e \in E$, there are at most $|e|(|e|-1)/2$ edges in G), we arrive at the total time complexity of DRSP.

For Gallo's Algorithm, similar to Dijkstra's algorithm, the time complexity is mainly in updating the neighbors of the non-fixed vertex z with the minimal distance $D[z]$. For each z, the algorithm scans all the hyperedges containing z. For each pair of vertices $(u, v) \in e$, e is scanned twice. Therefore, the total number of such operations is $\Phi = \sum_{e \in E} |e|^2$. Also, extracting z from the priority queue implemented by a Fibonacci heap takes $O(\log n)$ time. The total time complexity of Gallo's algorithm thus follows.

For a simplicial complex, $\Phi = O(d^2 2^d m)$, the complexity of Gallo's algorithm thus follows. For DR-SP, exploiting the property that the edge set is closed under the subset operation in a simplicial complex, we can use a top-down scheme in Step 1 of DR-SP to calculate the weight $w_G(s)$ inductively with respect to the dimension of a facet as follows:

$$w_G(s) = \min\{w(s), \{w_G(s') |\ s' \supset s,\ \dim[s'] = i + 1\}\}$$

where $w_G(s') = w(s')$ for the facet s'. The time complexity for Step 1 can then be improved. Because each i-dimensional face is associated with $d - i$ comparisons, the running time of Step 1 for each d-dimensional facet is given by

$$\sum_{i=1}^{d-1} \binom{d+1}{i+1} (d - i) = O(d2^d).$$

Therefore, the time complexity for Step 1 is $O(d2^d m)$. The total time complexity thus follows. ∎

From Theorem 4, we see that if δ is small and $|\check{e}|$ is large, HE-DSP performs better, since in DR-DSP, the induced graph has to be updated regardless of whether there are affected vertices. Thus, in a sequence of hyperedge changes, if only a small fraction of them actually have affected nodes, then HE-DSP will outperform DR-DSP. On the other hind, if δ is large, DR-DSP will outperform HE-DSP since usually $|\delta_\Phi| \gg \|\tilde{\delta}\|$.

VI. Directed Hypergrpah

In this section, we extend both algorithms to directed hypergraphs.

A. Definitions

In a directed hypergraph $H = (V, E)$, each hyperedge $e \in E$ consists of a source vertex S_e and a set of terminal vertices T_e (see Fig. 3 for an illustration). A directed hyperpath L from s to t is a sequence of directed hyperedges $L = \{e_1, e_2, \ldots, e_k\}$ such that $S_{e_1} = s$, $t \in T_{e_k}$ and $S_{e_i} \in T_{e_{i-1}}$ for $i = 2, \ldots, k$. See Fig. 3 for an example of a hyperpath with two hyperedges: $L = \{e_1, e_2\}$ where $S_{e_1} = 1, T_{e_1} = \{2, 3, 4, 5\}, S_{e_2} = 3, T_{e_2} = \{4, 6, 7\}$.

Based on the definition, one can easily see that an undirected hypergraph can be seen as a special case of a directed hypergraph. Specifically, each undirected hyperedge e with k vertices can be treated as k directed hyperedges, each from a source vertex v in e to the destination set $e - v$. Additionally, these k directed hyperedges share the same weight as the undirected hyperedge.

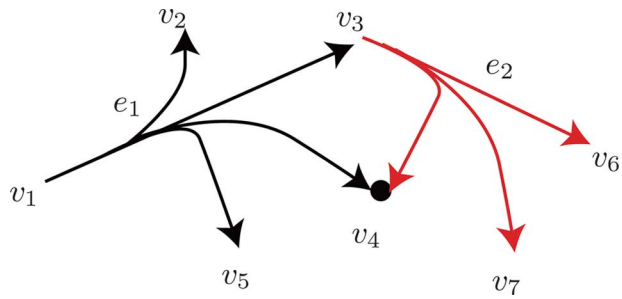

Fig. 3. Directed hyperpath with two directed hyperedges.

B. Extensions

First, we extend HE-DSP to directed hypergraphs. Consider first the case with weight decrease. In Step 1, we still identify the affected vertices in the changed hyperedge e. However, since the changed hyperedge has only one source vertex, if any other vertex is affected, its new shortest path has to come from the source vertex. Specifically, if a vertex $u \in T_e$ satisfies $D[S_e] + w_{\text{new}} < D[u]$, then it has a shortest path from S_e through the changed hyperedge e and becomes an affected vertex. We then put it into the priority queue Q. In Step 2, we still extract the vertex z with minimal current distance in Q and use it to update neighbors. Since the model is directed, instead of looking at all hyperedges that contain z, we only consider those directed hyperedges sourced at z. The rest of the algorithm remains the same.

For weight increase, the extension of HE-DSP is similar to the above process. In Step 1, we put all the receiving vertices in M to initialize the coloring. Then, in both coloring and updating process, when checking or updating the state of neighbors, we simply change the set of neighbors through one hyperedge e to the receiving set T_e.

Next, we consider an extension of the DR-DSP algorithm. Since the original hypergraph H is directed, the induced graph G becomes a directed graph. The weight of one edge $u \to v$ is the minimum weight among all hyperedges that provide a path from u to v. More specifically, the weight $w_{u,v}$ is given by

$$w_{u,v} = \min_{e \in E: u = S_e, v \in T_e} w(e). \tag{3}$$

Theorem 3 can be easily extended to the directed case. Hence, the problem is reduced to a special batch dynamic shortest path problem in directed graph, and both DR-DSP: Weight Decrease and DR-DSP: Weight Increase can be extended.

VII. Simulation Examples

We present simulation results on the running time of the proposed dynamic shortest hyperpath algorithms. We test the proposed algorithms on hypergraphs generated from a random geometric model as well as those generated by the Enron e-mail data set. All simulation code is compiled and run on the same laptop equipped with a 3.0-GHz i7–920XM Mobile Processor.

A. Random Geometric Hypergraph

We first consider a random geometric hypergraph model in which n nodes are uniformly distributed in an $a \times a$ square. All

nodes within a circle with radius r form a hyperedge (circles are centered on an $h \times h$ grid). The weight of each hyperedge is given by the average distance between all pairs of vertices of this hyperedge.

A sequence of changes is then generated, and the proposed dynamic algorithms are employed to maintain all the shortest hyperpaths from the source s located at a corner of the $a \times a$ square. Each change can be a hyperedge insertion (with probability $p_{\rm I}$), a hyperedge deletion (with probability $p_{\rm D}$), or a weight change (with probability $1 - p_{\rm I} - p_{\rm D}$) with new weight chosen uniformly in $[w_{\min}, w_{\max}]$. In the case of a hyperedge deletion or a weight change, the hyperedge to be deleted or to be assigned a new weight is chosen according to the two models detailed below. Hyperedge insertions are only *realized* when there are hyperedges that have been deleted, and a randomly chosen one is inserted back. This ensures that all hyperedges satisfy the geometric property determined by r at all time. It also models the practical scenario where a broken link is repaired.

In selecting a hyperedge for deletion or weight change, we consider two different models: the random change model and the targeted change model. In the former, the hyperedge is randomly and uniformly chosen among all hyperedges. In the latter, it is randomly and uniformly chosen from the current shortest hyperpaths. This models the scenarios where hyperedges in the shortest hyperpaths are more prone to changes due to attacks, frequent use, or higher priority in maintenance and upgrade.

In Fig. 4, we show the simulation results on the running time of the two proposed algorithms under a sequence of 10^4 changes. We see that HE-DSP has lower complexity in networks with random topological and weight changes (Fig. 4, top), whereas DR-DSP should be preferred in networks with targeted changes (Fig. 4, bottom). This partition of the application space can be explained from the structures of these two algorithms. Under the random change model, a large fraction of changes does not result in changes in the current shortest hyperpaths. Such changes lead to little computation in maintaining the shortest hyperpaths for both algorithms, but require about the same amount of computation in the Graph-Update step of DR-DSP for maintaining the induced graph. On the other hand, under the targeted change model, all hyperedge deletions and weight changes affect the shortest hyperpaths. Updating the shortest hyperpaths can be done more efficiently in DR-DSP since it works on the induced graph with a much smaller number of edges.

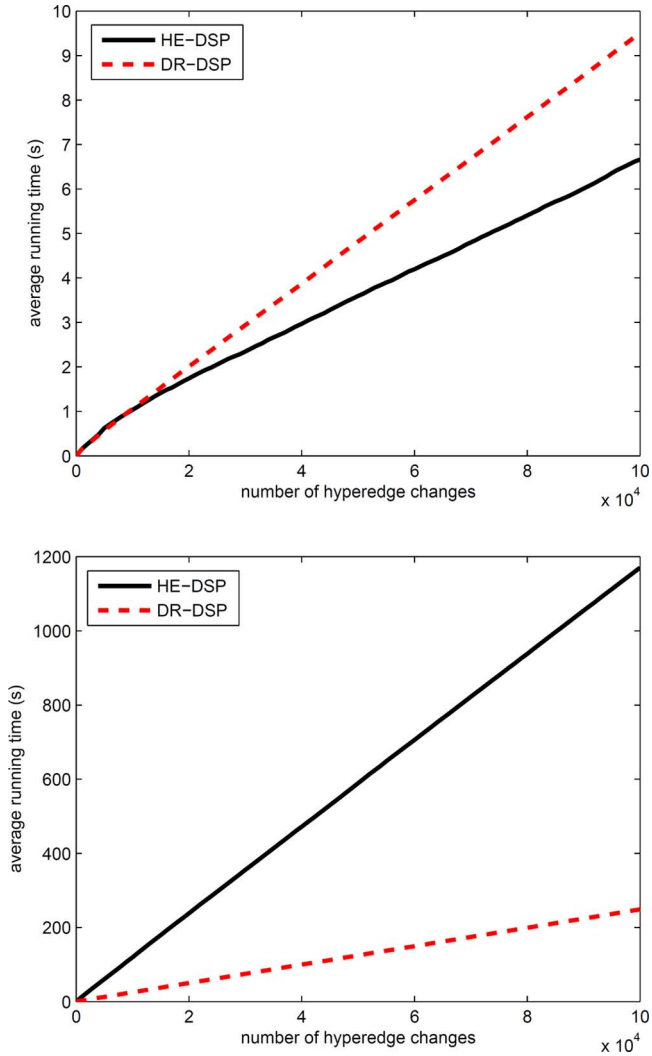

Fig. 4. Average running time. *(top)* Random change model. *(bottom)* Targeted change model ($n = 1000$, $a = 1000$, $r = \sqrt{1000}$, $h = 1$, $p_{\rm I} = \frac{1}{4}$, $p_{\rm D} = \frac{1}{4}$, $w_{\min} = 10$, $w_{\max} = 20$, the average is taken over 50 random hypergraphs).

B. Energy-Efficient Routing in Multichannel Multiradio Networks

Consider a multichannel multiradio ad hoc network with n nodes and m channels. Each node i is associated with a fixed transmission range R_i, a transmission energy cost α_i, a receiving energy cost β_i, and a list of $m_i \leq m$ channels $\{c_{k_1}, c_{k_2}, \ldots, c_{m_i}\} \subseteq \{c_1, c_2, \ldots, c_m\}$ available to it.

This network can be modeled by a directed hypergraph where a directed hyperedge e consists of one source vertex S_e and a set of destination vertices T_e. Specifically, each node i is a vertex v_i in the hypergraph. For each channel c_j available to node v_i, there is a hyperedge e_{ij} from v_i to the set V_{ij} of vertices consisting of nodes that are within the transmission range of v_i and share the channel c_j (if V_{ij} is empty, we do not consider this hyperedge). Then, the cost of this hyperedge is defined as

$$w(e_{ij}) = \alpha_i + \sum_{v \in V_{ij}} \beta_j. \tag{4}$$

We consider a mobile network. In this case, all four types of hyperedge changes (i.e., edge insertion, edge deletion, weight increase, and weight decrease) can occur, and multiple changes can occur simultaneously. We thus have a fully dynamic batch problem (as stated earlier, the proposed algorithms can handle batch changes with minor changes in their implementations).

In the simulation results shown in Fig. 5, nodes are uniformly distributed within an $a \times a$ square. The list of channels available to each node is drawn from all channels with probability p. At each time, one node is randomly chosen to move. The movement $(\Delta x, \Delta y)$ is generated by a random vector $(\Delta X, \Delta Y)$. We have implemented five algorithms: repeating static algorithm, repeating single change HE-DSP, repeating single change DR-DSP, batch HE-DSP, and batch DR-DSP. As we can see from Fig. 5, all dynamic algorithms outperform the

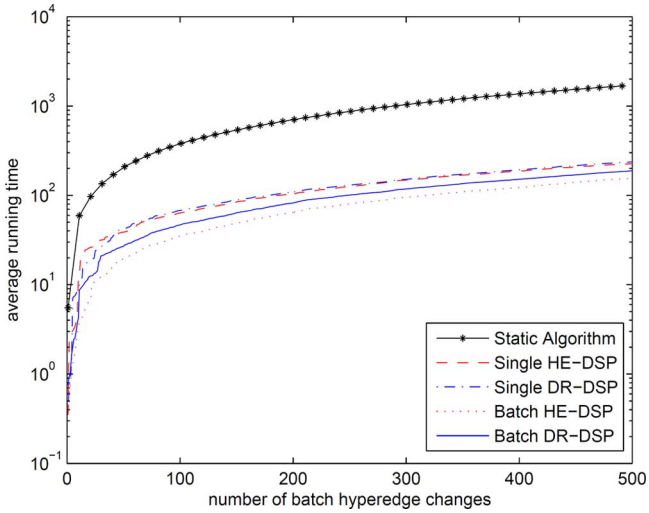

Fig. 5. Average running time with parameters $n = 1000$, $m = 20$, $a = \frac{n}{2}$, $R \sim U(0.8, 1.2)$, $\alpha \sim U(8, 12)$, $\beta \sim U(0.8, 1.2)$, $\Delta X, \Delta Y \sim U(-0.5, 0.5)$, and $p = 0.2$. The average is taken over 50 random hypergraphs.

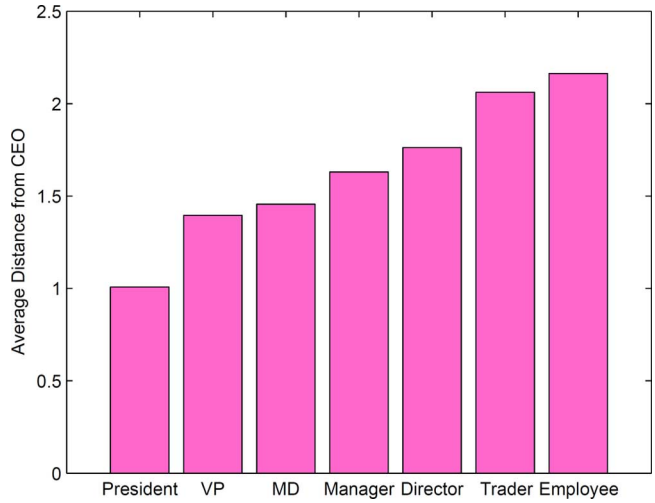

Fig. 6. Average distance from the CEO to others at different positions.

static algorithm. Also, the batch algorithms perform better than repeating the single change dynamic algorithms. In both single and batch algorithms, HE-DSP are better than DR-DSP. This is mainly because many hyperedge changes occur outside the current shortest hyperpaths.

C. Enron E-Mail Data Set

In this example, we consider the application of the shortest hyperpath algorithms in finding the most important actor in a social network. We consider the Enron e-mail data set and use the same hypergraph generation model as in [15]. Specifically, each person is a vertex of the hypergraph, and the sender and recipients of every e-mail form a hyperedge. Our objective is to identify the most important person measured by the closeness centrality index (i.e., the total weight of the shortest hyperpaths from this person to all the other persons). The first step is to assign weight to each hyperedge that reflects "distance." While there is no universally accepted way of measuring distance in a social network observed through e-mail exchanges, certain general rules apply. First, a direct e-mail exchange between two persons indicates a stronger tie than an e-mail sent to a large group. Thus, the weight of a hyperedge should be an increasing function of the cardinality of this hyperedge. Second, more frequent e-mail exchange among a given group of people shows stronger ties. Thus, the weight of a hyperedge should be decreasing with the number of times that this hyperedge appears in the e-mail data set. Considering these two general rules, we adopt the following weight function:

$$w(e) = (\sqrt{|e|})^{\alpha^{(l-1)}} \tag{5}$$

where $|e|$ is the cardinality of the hyperedge e, and α is the parameter measuring how fast the weight decreases with the number l of times that this hyperedge appears in the data set.

We can then apply DR-SP on the resulting (static) hypergraph to find the shortest hyperpaths rooted at each vertex and compute this vertex's closeness centrality index. With the weight function given in (5) using $\alpha = 0.6$, the identified most important actor is the CEO of Enron. The average distance (along the shortest hyperpath) from the CEO to other persons at various positions is shown in Fig. 6. We observe that, in general, the higher the position, the shorter the distance. These results demonstrate that the adopted hypergraph model and weight function capture the essence of the problem.

To demonstrates advantage of the hypergraph model, we run the same process based on the following graph model. Each person is a vertex of the graph. There exists an edge between two vertices u, v only if there is an e-mail that involves the two people represented by u, v either as sender or receiver. In comparison to the hypergraph model, we substitute $|e|$ in (5) to 2, the cardinality of an edge, in the weight function

$$w(u, v) = (\sqrt{2})^{\alpha^{(l-1)}}. \tag{6}$$

In this model, the weight of an edge only depends on the frequency of e-mail exchanges, but not the number of people involved in those e-mails. First, we use Dijkstra's algorithm to compute closeness centrality for each vertex. Based on the results, the most important actor is still identified to be the CEO of Enron. However, the average distances from the CEO no longer reveal the hierarchy of this company for the entire range of α (values from 0.1 to 0.9 with a step size of 0.1 were tested). Fig. 7 shows the result for $\alpha = 0.6$. The result is similar for other values of α.

Next, we construct a dynamic hypergraph sequence based on the Enron data set. At the beginning, the hypergraph contains only individual vertices. We then consider each e-mail chronologically. Each e-mail either adds a new hyperedge or decreases the weight of an existing hyperedge (due to the increased number of appearances of this hyperedge). The two proposed algorithms are employed to maintain the shortest hyperpaths rooted at the CEO after each change. The running time is given in Fig. 8, which shows the lower complexity of DR-DSP.

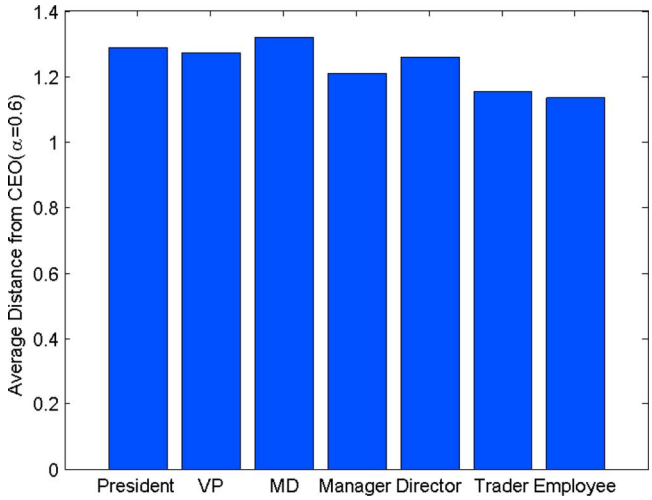

Fig. 7. Average distance from the CEO to others at different positions obtained in the graph model ($\alpha = 0.6$).

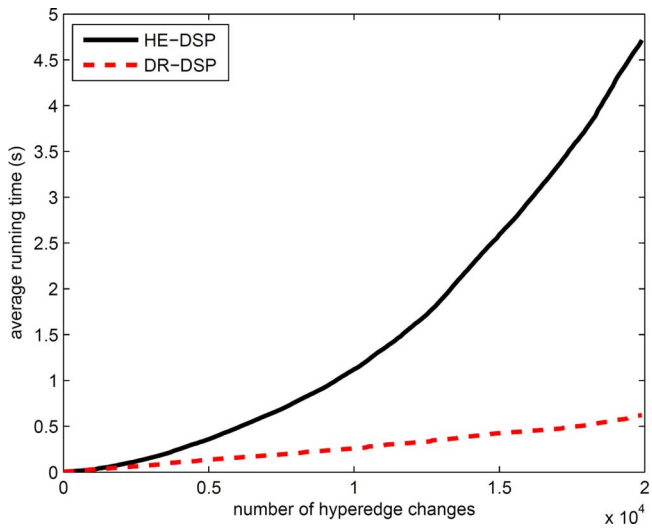

Fig. 8. Average running time for the Enron data set ($\alpha = 0.6$, the average is taken over 50 Monte Carlo runs).

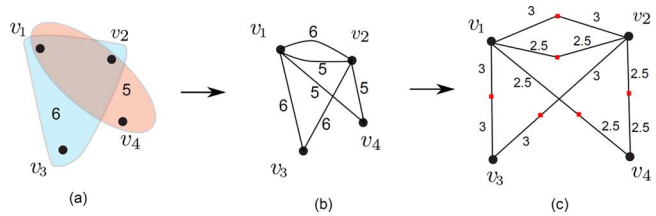

Fig. 9. Given a hypergraph (a), we first create a multigraph (b) by replacing each hyperedge with a clique whose edges have the same weight as the hyperedge. Then, each edge in this multigraph is replaced with one vertex and two edges to create a traditional graph (c). The weight of a new edge in the graph is half the weight of the multigraph edge from which the new edge is derived. As can be seen, the shortest path in the induced graph represents the shortest path in the original hypergraph. Maintaining this underlying graph after one hyperedge change takes $O(n^2)$ time. However, the number of vertices in the induced graph is more than the sum of the number of vertices and the number of hyperedges in the original hypergraph.

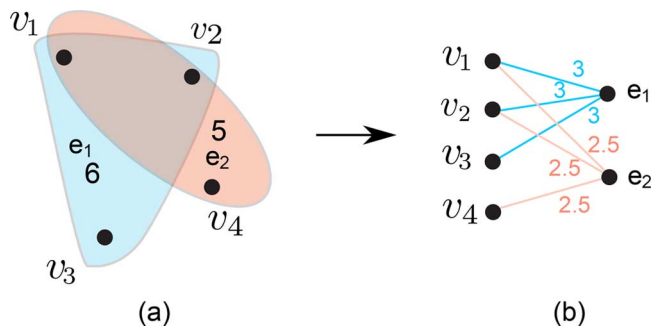

Fig. 10. Given a hypergraph H (a), we create its dual-transformation bipartite-graph G (b): Each vertex on the left side of G represents a vertex in the H, and each vertex on the right side represents a hyperedge in H. There is an edge between two vertices if the corresponding hyperedge contains the corresponding vertex in H. The weight of the edge in G is half the weight of the hyperedge represented by the right vertex of this edge. It is easy to see that the shortest path problem in H is equivalent to the shortest path in G. Maintaining this underlying graph after one hyperedge change takes $O(n)$ time. However, the number of vertices in the induced graph equals the sum of the number of vertices and the number of hyperedges in the original hypergraph.

The reason is that a large fraction of hyperedge changes result in changes in the shortest hyperpaths.

VIII. Discussion

The shortest path problem in a hypergraph can be transformed to the shortest path problem in an induced graph, and there are many ways of transforming the problem. As an algorithmic study of the problem where the objective is the efficiency of the solution, we aim to address the following two questions in this paper: 1) When is it more efficient to transform the problem into a graph problem rather than working directly on the hypergraph? 2) When it is more efficient to transform the problem, which graph transformation has better time complexity?

The first question was addressed by a detailed complexity comparison, both analytically (Section V) and empirically (Section VII), of the two proposed algorithms, one working directly on the hypergraph, the other based on a graph transformation. This complexity comparison leads to a partition of the application domain for these two approaches based on the type of the hypergraph and the nature of the dynamics of the weight changes.

To tackle the second question, we first notice that to capture all the information of an edge-weighted hypergraph with n vertices, the induced graph can be exponentially large with n since the number of hyperedges (hence hyperedge weights) can be exponential with n. Figs. 9 and 10 demonstrate two examples of a graph transformation that can result in an exponentially large graph. Since time complexity of dynamic shortest path algorithms can be the same as a static one in the worst case [2], [4], [5], applying them to these types of underlying graph that represents complete information of the given hypergraph results in exponential time complexity with n

When adapting a graph reduction-based approach, we do need to maintain the complete information of the hypergraph in order to update the hypergraph for future hyperedge changes. However, we also want to reduce the size of the underlying graph so that applying a dynamic shortest path algorithm to it results in polynomial time complexity with n in the worst case.

TABLE I
COMPARISON BETWEEN THREE GRAPH-TRANSFORMATION-BASED APPROACHES

	Multi-graph approach in Fig. 9	Bipartite graph approach in Fig. 10	DR-DSP
Number of Vertices in G	$\Theta(2^n)$	$\Theta(2^n)$	n
Maintaining G	$O(n^2)$	$O(n)$	$O(n^3)$
Applying DSP to G	$O(2^n)$	$O(2^n)$	$O(n^2)$
Total Time Complexity	$O(2^n)$	$O(2^m)$	$O(n^3)$

Comparison of the three graph-reduction based approaches while there are $\Theta(2^n)$ hyperedges in the hypergraph and one hyperedge change causes all vertices affected. The first row is the number of vertices in the resulting graph G after the transformation. The second row is the time complexities for maintaining the underlying graphs. The third row is the time complexities of applying dynamic shortest path algorithms to the underlying graph G. The last row is the total time of the corresponding graph-transformation based approach.

We overcome this dilemma in the proposed DR-DSP algorithm by constructing a two-level data structure. One level is a list of priority queues for maintaining the complete information of the hypergraph. The other level is an underlying graph with n vertices that only contains the essential information needed for the shortest path calculation. Though the size of the data structure at the first level can be exponential with n, maintaining it and the underlying graph after each hyperedge change only requires polynomial time complexity with n (specifically, $O(n^3)$ in the worst case). Since the underlying graph has only n vertices, updating the shortest paths after each change also has a time complexity polynomial with n. Table I gives a detailed complexity comparison of the proposed DR-DSP algorithm with the two alternative graph-transformation-based approaches given in Figs. 9 and 10. It shows that DR-DSP reduces the time complexity from exponential to polynomial by striking a balance between the complexity for maintaining the resulting graph G after the transformation and the complexity for maintaining the shortest paths in G.

IX. CONCLUSION

We have presented the first study of the fully dynamic shortest path problem in a general hypergraph. We have developed two dynamic algorithms for finding and maintaining the shortest hyperpaths. These two algorithms complement each other with each one preferred in different types of hypergraphs and network dynamics, as illustrated in the time complexity analysis and simulation experiments. The special two-level data structure of our second algorithm, DR-DSP, may also find other applications in dynamic hypergraph studies. We have discussed and studied via experiments over a real data set the potential applications of the dynamic shortest hyperpath problem in social and communication networks.

APPENDIX A
PROOF OF THEOREM 1

The proof is based on the following three lemmas.

Lemma 2: Let $x = \arg\min_{v\in\check{e}}\{d[v]\}$, then $d[x] = d'[x]$ and $d'[x] = \min_{v\in\check{e}}\{d'[v]\}$.

Proof: Proof by contradiction. Assume that $d'[x] < d[x]$, then x has to use $\check{e}$ on its new shortest hyperpath. Since we consider only simple hyperpaths and $x \in \check{e}$, we have $E[x] = \check{e}$. Therefore, its parent $y = P[v]$ cannot use $\check{e}$ on its shortest hyperpath, which implies that the shortest distance to y does not change: $d[y] = d'[y]$. Given that y is the parent of x on its new shortest hyperpath, we have $d[y] = d'[y] \le d'[x] < d[x]$, which contradicts the definition of x.

For the second statement, assume that there exists $z \in \check{e}$ such that $d'[z] < d'[x]$. Based on the definition of x and the hypothetical assumption, $d[z] \ge d[x] = d'[x] > d'[z]$. It thus follows that z's shortest hyperpath changes and $E[z] = \check{e}$ in the new shortest hyperpath. Following the same line of arguments by considering the parent of z, we arrive at the same contradiction in terms of the definition of x. ■

Lemma 3: For any vertex v, v is enqueued into Q if and only if $d'[v] < d[v]$.

Proof: Consider first that v is enqueued into Q. From the algorithm, this can only happen if there exists a neighbor z and a hyperedge $e \ni v, z$ such that $D[z] + w(e) < D[v]$. We thus have $d[v] \ge D[v] > D[z] + w(e) \ge D'[v]$ (note that at any time, $d[v] \ge D[v] \ge D'[v]$, which can be easily seen from the procedure of the algorithm).

We now prove the converse. Assume that $d'[v] < d[v]$. Let $p = \{e_1, e_2, \ldots, e_i, \check{e}, e_{i+1}, \ldots, e_l\}$ be v's new shortest hyperpath. There exists $u_{i+1} \in \check{e} \cap e_{i+1}$ such that $d'[u_{i+1}] < d[u_{i+1}]$. In Step 1 of the algorithm, u_{i+1} is enqueued. Similarly, there exists $u_{i+2} \in e_{i+1} \cap e_{i+2}$ with $d'[u_{i+2}] < d[u_{i+2}]$. Then, u_{i+2} will be enqueued in Step 2 of the algorithm when u_{i+1} is dequeued if it has not been enqueued before that. Repeating this line of argument, we conclude that there exits $u_l \in e_{l-1} \cap e_l$ with $d'[u_l] < d[u_l]$ and u_l is enqueued into Q. Then, v will be enqueued when u_l is dequeued if it is not enqueued already. ■

Lemma 4: For each v dequeued from Q, $D[v] = d'[v]$.

Proof: We first show that if u is dequeued before v, then $D[u] \le D[v]$ at the instants when they are dequeued. We prove this by induction. The initial condition holds trivially. Then, assume it is true for the first l dequeued vertices $z_1, \ldots, z_l$. Consider the $(l+1)$th dequeued vertex z_{l+1}. At the instant when z_l is dequeued, if $D[z_{l+1}]$ is updated based on $D[z_l]$ in Step 2, then $D[z_l] < D[z_{l+1}]$ even after the update. If, on the other hand, $D[z_{l+1}]$ is not updated at this instant, then $D[z_l] \le D[z_{l+1}]$ given that the dequeued vertex has the smallest distance.

Next, we prove the lemma by induction. From Step 1 of the algorithm, all the affected vertices v in $\check{e}$ will be dequeued first with $E[v] = \check{e}$, $P[v] = x$, and $D[v] = d'[x] + w(\check{e})$. Based on Lemma 2, $D[v] \le d'[u] + w(\check{e})$ for any $u \in \check{e}$. It thus follows that the hyperpath to v through x and $\check{e}$ is the shortest one with $D[v] = d'[v]$.

Assume for $z_1, \ldots, z_l$, $D[z_i] = d'[z_i]$ are satisfied for all $i = 1, \ldots, l$. Consider the $(l+1)$th dequeued vertex $z_{l+1} \notin \check{e}$. Let $u = P[z_{l+1}]$ be its parent in the new shortest hyperpath. Then, based on the fact that distances of the dequeued vertices are monotonically increasing with the order of the dequeueing as shown at the beginning of the proof, u cannot be any vertex dequeued after z_{l+1}. Since $z_{l+1} \notin \check{e}$, it is also clear that u cannot

be an unaffected vertex (otherwise, z_{l+1} will be unaffected, which contradicts Lemma 3). We thus have $u \in \{z_1, \ldots, z_l\}$. Let $u = z_i$. Then, when z_i is dequeued, $D[z_{l+1}]$ will be updated to the shortest distance $d'[z_{l+1}]$ due to the induction hypothesis of $D[z_i] = d'[z_i]$. This completes the proof. ■

Based on Lemmas 3 and 4, the shortest distances of all affected vertices will be updated correctly. Based on Lemma 3, no unaffected vertex will be enqueued, and their distances remain the same. It is not difficult to see from the algorithm that $P[v]$ and $E[v]$ are also correctly maintained for all v.

Appendix B
Proof of Theorem 2

We first show the correctness of the coloring process as given in the following lemma.

Lemma 5: The coloring process correctly colors all the affected vertices.

Proof: We first state the following simple facts without proof: Given a relationship tree, after the hyperedge weight increase: 1) if v is pink or white, then all its descendent in this relationship tree are white; 2) if a v is red, then all its children in the relationship tree are either pink or red; 3) if v is affected, either $v \in \check{e}$ or $P[v]$ is red. These facts can be directly obtained from the definition of the colors. It is also easy to see that vertices are dequeued from M in a nondecreasing order of their current distance $D[\cdot]$. This is because each time a vertex z is dequeued from M, the possible new vertices to be enqueued into M are z's children with distances no smaller than $D[z]$.

Then, the proof of the lemma has two parts: First, we prove that all affected vertices are enqueued into M; then we prove by induction that only affected vertices are enqueued into M and their colors are correctly identified.

We prove the first part by contradiction. Assume that there exists an affected vertex v that is not enqueued into M. It is easy to see that $v \notin \check{e}$ because all the affected vertices in $\check{e}$ are enqueued in Step 1. Based on the third fact stated above, $P[v]$ is red. Based on the hypothesis, $P[v]$ is not enqueued (otherwise, v will be enqueued in Step 2). Continuing this line of arguments, we eventually reach the root of the relationship tree and arrive at the contradiction that the source s is red.

We prove the second part by induction. It is easy to see that all the vertices initially enqueued into M are affected vertices. It remains to show that the first vertex z_1 dequeued from M is colored (pink or red) correctly. To show that, we need to establish that the algorithm correctly determines whether there is an alternative shortest hyperpath to z_1 with the same distance, i.e., $d[z_1] = d'[z_1]$. The key here is to show that checking the currently nonred neighbors (which may become red in the future) of z_1 will not lead to a false alternative path. This follows from the fact that z_1 has the smallest distance $D[\cdot]$ among all affected vertices (which belong to the set of vertices consisting of the affected vertices in $\check{e}$ and their descendents).

Next, assume that vertices $z_1, z_2, \ldots, z_l$ dequeued from M are all affected vertices and are correctly colored. Consider the next dequeued vertex z_{l+1}. It is an affected vertex because it is either enqueued in Step 1 with $E[v] = \check{e}$ or enqueued in Step 2 with a red parent. To show that z_{l+1} will be colored correctly, we use a similar argument by showing that the currently nonred neighbors of z_{l+1} will not give a false alternative path. The latter follows from the fact that all affected vertices will be enqueued and those dequeued after z_{l+1} have distances no smaller than $D[z_{l+1}]$. This completes the induction. ■

We now show that $D[v]$, $P[v]$ and $E[v]$ are correctly maintained for all v. For each red vertex v, its distance is set based on the current shortest distance from a nonred neighbor in Step 3.a. The rest of the algorithm is essentially Gallo's extension of Dijkastra's algorithm with the current initial distance. The correctness of the algorithm thus follows. It is not difficult to see that $P[\cdot]$ and $E[\cdot]$ are correctly updated for both red and pink vertices.

References

[1] C. Berge, *Graphs and Hypergraphs*. Amsterdam, The Netherlands: North-Holland, 1976.

[2] G. Ramalingam and T. Reps, "On the computational complexity of dynamic graph problems," *Theoret. Comput. Sci.*, vol. 158, no. 1–2, pp. 233–277, 1996.

[3] G. Ramalingam and T. Reps, "An incremental algorithm for a generalization of the shortest-path problem," *J. Algor.*, vol. 21, no. 2, pp. 267–305, 1996.

[4] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni, "Fully dynamic algorithms for maintaining shortest paths trees," *J. Algor.*, vol. 34, no. 2, pp. 251–281, 2000.

[5] P. Narvaez, K. Siu, and H. Tzeng, "New dynamic algorithms for shortest path tree computation," *IEEE/ACM Trans. Netw.*, vol. 8, no. 6, pp. 734–746, Dec. 2000.

[6] G. Gallo, G. Longo, S. Nguyen, and S. Pallottino, "Directed hypergraphs and applications," *Discrete Appl. Math.*, vol. 42, no. 2–3, pp. 177–201, Apr. 1993.

[7] G. Ausiello, G. Italiano, and U. Nanni, "Optimal traversal of directed hypergraphs," Int. Comput. Sci. Inst., 1992.

[8] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *Proc. 10th Annu. Mobicom*, 2004, pp. 114–128.

[9] S. Muruganathan, D. Ma, R. Bhasin, and A. Fapojuwo, "A centralized energy-efficient routing protocol for wireless sensor networks," in *Proc. IEEE Radio Commun.*, Mar. 2005, pp. 8–13.

[10] C. Schurgers and M. Srivastava, "Energy efficient routing in wireless sensor networks," in *Proc. IEEE Military Commun. Conf.*, 2001, vol. 1, pp. 357–361.

[11] K. Scott and N. Bambos, "Routing and channel assignment for low power transmission in PCS," in *Proc. 5th IEEE Int. Conf. Universal Pers. Commun.*, 1996, vol. 2, pp. 498–502.

[12] V. Rodoplu and T. Meng, "Minimum energy mobile wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 8, pp. 1333–1344, Aug. 1999.

[13] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*. New York, NY, USA: Cambridge Univ. Press, 1997.

[14] N. B. Dale, *C++ Plus Data Structures*. Sudbury, MA, USA: Jones & Bartlett, 2006.

[15] D. M. Y. Park, C. E. Priebe, and D. J. Marchette, "Scan statistics on Enron hypergraphs," in *Proc. Interface*, 2008.

Jianhang Gao received the Bachelor's degree in engineering physics from Tsinghua University, Beijing, China, in 2010, and is currently pursuing the Ph.D. degree in electrical and computer engineering at the University of California, Davis, CA, USA.

He joined the Signal Processing and Adaptive Networking Group, University of California, Davis, under instruction of Prof. Qing Zhao in 2010, researching in the area of algorithmic study of hypergraph and simplicial complex.

Qing Zhao (S'97–M'02–SM'08–F'12) received the Ph.D. degree in electrical engineering from Cornell University, Ithaca, NY, USA, in 2001.

In 2004, she joined the Department of Electrical and Computer Engineering, University of California, (UC)Davis, CA, USA, where she is currently a Professor. She is also a Professor with the Graduate Group of Applied Mathematics, UC Davis. Her research interests are in the general area of stochastic optimization, decision theory, and algorithmic theory in dynamic systems and communication and social networks.

Dr. Zhao received the 2010 *IEEE Signal Processing Magazine* Best Paper Award and the 2000 Young Author Best Paper Award from the IEEE Signal Processing Society. She holds the title of UC Davis Chancellors Fellow and received the 2008 Outstanding Junior Faculty Award from the UC Davis College of Engineering. She was a plenary speaker at the 11th IEEE Workshop on Signal Processing Advances in Wireless Communications (SPAWC), 2010. She is also a coauthor of two papers that received student paper awards at ICASSP 2006 and the IEEE Asilomar Conference 2006.

Wei Ren received the B.Sc. and M.Sc. degrees from Peking University, Beijing, China, in 2003 and 2006, respectively, and the Ph.D. degree from University of California, Davis, CA, USA, in 2011.

He is currently a Software Development Engineer with Microsoft Corporation, Redmond, WA, USA. During the Ph.D. study, his research interests were in cognitive radio systems and wireless networks. He was also interested in algorithmic theory and optimization techniques for communications.

Ananthram Swami (S'79–M'79–SM'96–F'08) received the B.Tech. degree from the Indian Institute of Technology (IIT), Bombay; India, the M.S. degree from Rice University, Houston, TX, USA, and the Ph.D. degree from the University of Southern California (USC), Los Angeles, CA, USA, all in electrical engineering.

He is with the U.S. Army Research Laboratory (ARL) as the Army's ST (Senior Research Scientist) for Network Science. He has held positions with Unocal Corporation, USC, CS-3, and Malgudi Systems. He was a Statistical Consultant with the California Lottery, developed a MATLAB-based toolbox for non-Gaussian signal processing, and has held visiting faculty positions with INP, Toulouse, France. His research interests are in the broad area of network science: the study of interactions and co-evolution, prediction and control of interdependent networks, with applications in composite tactical networks.

Dr. Swami is an ARL Fellow.

Ram Ramanathan (S'92–M'92–SM'97–F'12) received the Bachelor of Technology degree in electrical engineering from the Indian Institute of Technology, Madras, India, in 1985, and the M.S. and Ph.D. degrees in computer and information sciences from the University of Delaware, Newark, DE, USA, in 1989 and 1993, respectively.

He is a Principal Scientist with the Network Research Department, Raytheon BBN Technologies, Cambridge, MA, USA, where he helps invent, develop, and prototype new technologies, primarily in wireless computer networks. His current research areas broadly include wireless networks, network analysis and control algorithms, and network science. Over the course of his career, he has made contributions to the areas of scalable routing, topology control, and the use of directional antennas in multihop networks and led several projects for the Department of Defense, in particular DARPA.

Dr. Ramanathan has served on the editorial boards of the IEEE TRANSACTIONS ON MOBILE COMPUTING, of which he was the Associate Editor-in-Chief, and *Ad Hoc Networks*. He has served on the program committees of several conferences including MobiCom, Mobihoc, and IEEE INFOCOM, and was the TPC Co-Chair of Mobicom 2007. He has published widely in international journals and conferences, including Best Paper Award-winning papers at IEEE MILCOM, IEEE INFOCOM, and ACM SIGCOMM.

Amotz Bar-Noy received the B.Sc. degree in mathematics and computer science (*summa cum laude*) and Ph.D. degree in computer science from the Hebrew University, Jerusalem, Israel, in 1981 and 1987, respectively.

He was a Post-Doctoral Fellow with Stanford University, Stanford, CA, USA, from 1987 to 1989. He was a Research Staff Member with the IBM Research Center, Yorktown Heights, NY, USA, from 1989 to 1996. He was an Associate Professor with the Electrical Engineering Department, Tel Aviv University, Tel Aviv, Israel, from 1996 to 2001. He was a Principal Technical Staff Member with AT&T Research Labs, Florham Park, NJ, USA, from 1999 to 2001. Since 2002, he has been a Professor with the Computer and Information Science Department, Brooklyn College, City University of New York (CUNY), Brooklyn, NY, USA, and with the Computer Science Department, Graduate Center, CUNY, Manhattan, NY, USA. He has published more than 80 refereed journal articles and more than 100 refereed conference and workshop articles. His field of expertise belongs to the theoretical computer science community and to the networking community. The scope of his research is to bridge the gap between these two communities.

Prof. Bar-Noy served as a program committee member for many conferences. He was an Editor for the IEEE TRANSACTIONS ON MOBILE COMPUTING and *Wireless Networks* journal. He served as a Guest Editor for two special issues—one for *Wireless Networks* and one for *Mobile Networking and Applications*. He served as a co-chair of the ALGOSENSORS 2012 Symposium. In 2011, he was awarded the Edsger W. Dijkstra Prize in Distributed Computing for being an author of an outstanding paper on the principles of distributed computing, whose significance and impact on the theory and/or practice of distributed computing has been evident for at least a decade.